***Gcd***: return b==0 ? a : gcd(b,a%b);

***Lcm***: return a / gcd(a,b) \* b;

***Pi***: const double pi = acos(-1.0);

***分数转小数***:输出a/b余数=a%b

While(c--){ys\*=10;输出ys/b;ys%=b;

***解线性方程***:

for(int k = 1; k <= n; k++)

for(int i = 1; i <= n; i++)

if(i != k){

double d = a[i][k] / a[k][k];

for(int j = k+1; j <= n+1; j++)

a[i][j] = a[i][j] - d \* a[k][j];

}

for(i:1~n) x[i] = a[i][n+1] / a[i][i];

***解线性方程2***:

X=M1/M; Y=M2/M; Z=M3/M;

M = afk+bgi+cej-cfi-bek-agj;

M1 = bgl+chj+dfk-…….;

***Inverse Matrix***

tmp = 1 / (ad-bc);

d\*tmp –b\*tmp

-c\*tmp a\*tmp

树的层次遍历

输入(11,L) (12,R) (5,)

输出5 11 12

// Trees on the level

#include<cstdio>

#include<cstdlib>

#include<cstring>

#include<vector>

#include<queue>

using namespace std;

const int maxn = 256 + 10;

struct Node{

bool have\_value;

int v;

Node\* left, \*right;

Node():have\_value(false),left(NULL),right(NULL){}

};

Node\* root;

Node\* newnode() { return new Node(); }

bool failed;

void addnode(int v, char\* s) {

int n = strlen(s);

Node\* u = root;

for(int i = 0; i < n; i++)

if(s[i] == 'L') {

if(u->left == NULL) u->left = newnode();

u = u->left;

} else if(s[i] == 'R') {

if(u->right == NULL) u->right = newnode();

u = u->right;

}

if(u->have\_value) failed = true;

u->v = v;

u->have\_value = true;

}

void remove\_tree(Node\* u) {

if(u == NULL) return;

remove\_tree(u->left);

remove\_tree(u->right);

delete u;

}

char s[maxn];

bool read\_input() {

failed = false;

remove\_tree(root);

root = newnode();

for(;;) {

if(scanf("%s", s) != 1) return false;

if(!strcmp(s, "()")) break;

int v;

sscanf(&s[1], "%d", &v);

addnode(v, strchr(s, ',')+1);

}

return true;

}

bool bfs(vector<int>& ans) {

queue<Node\*> q;

ans.clear();

q.push(root);

while(!q.empty()) {

Node\* u = q.front(); q.pop();

if(!u->have\_value) return false;

ans.push\_back(u->v);

if(u->left != NULL) q.push(u->left);

if(u->right != NULL) q.push(u->right);

}

return true;

}

int main() {

vector<int> ans;

while(read\_input()) {

if(!bfs(ans)) failed = 1;

if(failed) printf("not complete\n");

else {

for(int i = 0; i < ans.size(); i++) {

if(i != 0) printf(" ");

printf("%d", ans[i]);

}

printf("\n");

}

}

return 0;

}

树：

// UVa548 Tree

// 題意：給一棵點帶權（權各不相同，都是正整數）二元樹的中序和後序走訪，找一個葉子使得它到根的路徑上的權和最小。如果有多解，該葉子本身的權應儘量小

// 演算法：遞迴建樹，然後DFS。注意，直接遞迴求結果也可以，但是先建樹的方法不僅直觀，而且更好除錯

#include<iostream>

#include<string>

#include<sstream>

#include<algorithm>

using namespace std;

// 因為各個節點的權值各不相同且都是正整數，直接用權值作為節點編號

const int maxv = 10000 + 10;

int in\_order[maxv], post\_order[maxv], lch[maxv], rch[maxv];

int n;

bool read\_list(int\* a) {

string line;

if(!getline(cin, line)) return false;

stringstream ss(line);

n = 0;

int x;

while(ss >> x) a[n++] = x;

return n > 0;

}

// 把in\_order[L1..R1]和post\_order[L2..R2]建成一棵二元樹，返回樹根

int build(int L1, int R1, int L2, int R2) {

if(L1 > R1) return 0; // 空樹

int root = post\_order[R2];

int p = L1;

while(in\_order[p] != root) p++;

int cnt = p-L1; // 左子樹的節點個數

lch[root] = build(L1, p-1, L2, L2+cnt-1);

rch[root] = build(p+1, R1, L2+cnt, R2-1);

return root;

}

int best, best\_sum; // 目前為止的最優解和對應的權和

void dfs(int u, int sum) {

sum += u;

if(!lch[u] && !rch[u]) { // 葉子

if(sum < best\_sum || (sum == best\_sum && u < best)) { best = u; best\_sum = sum; }

}

if(lch[u]) dfs(lch[u], sum);

if(rch[u]) dfs(rch[u], sum);

}

int main() {

while(read\_list(in\_order)) {

read\_list(post\_order);

build(0, n-1, 0, n-1);

best\_sum = 1000000000;

dfs(post\_order[n-1], 0);

cout << best << "\n";

}

return 0;

}

输入：3 2 1 4 5 7 6

3 1 2 5 6 7 4

输出：1

天平

// UVa839

// 題意：輸入一個樹狀天平，根據力矩相等原則判斷是否平衡。採用遞迴方式輸入，0表示中間節點

// 演算法：在"建樹"時直接讀入並判斷，並且無須把樹保存下來

#include<iostream>

using namespace std;

// 輸入一個子天平，返回子天平是否平衡，參數W修改為子天平的總重量

bool solve(int& W) {

int W1, D1, W2, D2;

bool b1 = true, b2 = true;

cin >> W1 >> D1 >> W2 >> D2;

if(!W1) b1 = solve(W1);

if(!W2) b2 = solve(W2);

W = W1 + W2;

return b1 && b2 && (W1 \* D1 == W2 \* D2);

}

int main() {

int T, W;

cin >> T;

while(T--) {

if(solve(W)) cout << "YES\n"; else cout << "NO\n";

if(T) cout << "\n";

}

return 0;

}

Play on Words

// 題意：輸入n個單字，是否可以排成一個序列，使得每個單字的第一個字母和上一個單字的最後一個字母相同

// 演算法：把字母看作節點，單字看成有向邊，則有解當且僅當圖中有尤拉路徑。注意要先判連通

#include<cstdio>

#include<cstring>

#include<vector>

using namespace std;

const int maxn = 1000 + 5;

int pa[256];

int findset(int x) { return pa[x] != x ? pa[x] = findset(pa[x]) : x; }

int used[256], deg[256]; // 是否出現過；度數

int main() {

int T;

scanf("%d", &T);

while(T--) {

int n;

char word[maxn];

scanf("%d", &n);

memset(used, 0, sizeof(used));

memset(deg, 0, sizeof(deg));

for(int ch = 'a'; ch <= 'z'; ch++) pa[ch] = ch; // 初始化並查集

int cc = 26; // 連通塊個數

for(int i = 0; i < n; i++) {

scanf("%s", word);

char c1 = word[0], c2 = word[strlen(word)-1];

deg[c1]++;

deg[c2]--;

used[c1] = used[c2] = 1;

int s1 = findset(c1), s2 = findset(c2);

if(s1 != s2) { pa[s1] = s2; cc--; }

}

vector<int> d;

for(int ch = 'a'; ch <= 'z'; ch++) {

if(!used[ch]) cc--; // 沒出現過的字母

else if(deg[ch] != 0) d.push\_back(deg[ch]);

}

bool ok = false;

if(cc == 1 && (d.empty() || (d.size() == 2 && (d[0] == 1 || d[0] == -1)))) ok = true;

if(ok) printf("Ordering is possible.\n");

else printf("The door cannot be opened.\n");

}

return 0;

}

油田

// 題意：輸入一個字元矩陣，統計字元@組成多少個四連塊

#include<cstdio>

#include<cstring>

const int maxn = 100 + 5;

char pic[maxn][maxn];

int m, n, idx[maxn][maxn];

void dfs(int r, int c, int id) {

if(r < 0 || r >= m || c < 0 || c >= n) return;

if(idx[r][c] > 0 || pic[r][c] != '@') return;

idx[r][c] = id;

for(int dr = -1; dr <= 1; dr++)

for(int dc = -1; dc <= 1; dc++)

if(dr != 0 || dc != 0) dfs(r+dr, c+dc, id);

}

int main() {

while(scanf("%d%d", &m, &n) == 2 && m && n) {

for(int i = 0; i < m; i++) scanf("%s", pic[i]);

memset(idx, 0, sizeof(idx));

int cnt = 0;

for(int i = 0; i < m; i++)

for(int j = 0; j < n; j++)

if(idx[i][j] == 0 && pic[i][j] == '@') dfs(i, j, ++cnt);

printf("%d\n", cnt);

}

return 0;

}

四分树

// Rujia Liu

// 題意：給兩棵四分樹的先序走訪，求二者合併之後（黑色部分合併）黑色像素的個數。p表示中間節點，f表示黑色（full），e表示白色（empty）

// 演算法：先建樹，然後統計

#include<cstdio>

#include<cstring>

const int len = 32;

const int maxn = 1024 + 10;

char s[maxn];

int buf[len][len], cnt;

// 把字串s[p..]匯出到以(r,c)為左上角，邊長為w的緩衝區中

// 2 1

// 3 4

void draw(const char\* s, int& p, int r, int c, int w) {

char ch = s[p++];

if(ch == 'p') {

draw(s, p, r, c+w/2, w/2); // 1

draw(s, p, r, c , w/2); // 2

draw(s, p, r+w/2, c , w/2); // 3

draw(s, p, r+w/2, c+w/2, w/2); // 4

} else if(ch == 'f') { // 畫黑像素（白像素不畫）

for(int i = r; i < r+w; i++)

for(int j = c; j < c+w; j++)

if(buf[i][j] == 0) { buf[i][j] = 1; cnt++; }

}

}

int main() {

int T;

scanf("%d", &T);

while(T--) {

memset(buf, 0, sizeof(buf));

cnt = 0;

for(int i = 0; i < 2; i++) {

scanf("%s", s);

int p = 0;

draw(s, p, 0, 0, len);

}

printf("There are %d black pixels.\n", cnt);

}

return 0;

}

Dijkstra

Memset(v,0,sizeof(v));

For(i:0~n-1) d[i]=i==0?0:INF

For(i:0~n-1){

Int x,m=INF;

For(y:0~n-1) if(!v[y]&&d[y]<=m)m=d[x=y]

V[x]=1;

For(y:0~n-1) d[y]=min(d[y],d[x]+w[x][y]

}